Visual Data Mining in Large-Scale 3D City Models

II International Conference and Exhibition on Geographic Information, Estoril Congress Center, May 30- June 2, 2005

Henrik BUCHHOLZ and Jürgen DÖLLNER

This paper presents an approach towards visual data mining in large-scale virtual 3D city models. The increasing availability of massive thematic data related to urban areas such as socio-demographic data, traffic data, or real-estate data, raises the question how to get insight and how to effectively visualize contained information. In our approach, we extend a real-time 3D city model system by features that interactively map thematic data to specified graphics variables of the city model's geometry and appearance. In particular, a rendering technique is explained that can efficiently represent and reconfigure the scene graph of a 3D city model even for large-scale models. The resulting dynamic 3D city models serve as general geovisualization tools to effectively analyze, explore, and present geometric and related thematic information of urban areas. Sample applications include city information systems, urban planning and management systems, and navigation systems.

KEYWORDS

Geovisualization, Information Visualization, 3D City Models, Data Mining, Information Retrieval.

INTRODUCTION

Virtual 3D city models have become an important issue for an increasing number of application areas. Since 3D city models are used for various purposes, the requirements made on their visualization vary between different applications. On the one hand, in the context of tourism, entertainment, or public participation a high degree of photorealism is required (Figure 1 left). For instance, if the aim is to give a realistic impression of a planned environment, the quality of a 3D visualization is directly related to the similarity between the virtual city model and the actual result after implementation of the planning. On the other hand, for certain applications visual details of buildings are not the primary interest. Instead, the 3D representation of a city model serves as a medium to convey spatial-related thematic information in a comprehensive way. In the context of urban planning, e.g., thematic building information such as vacancy, ownership, or year of construction have to be considered. As an example, the illustration in Figure 1 right, created by the Senate Department of Urban Development, Berlin, shows a 3D overview of the ownership structure

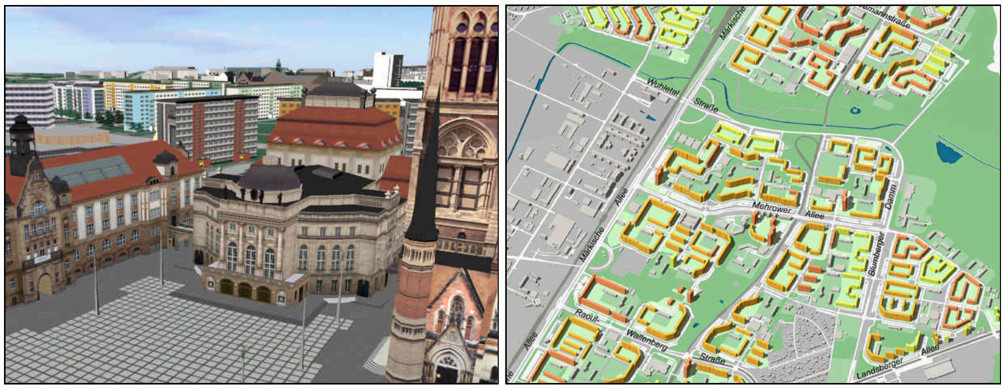

Figure 1: Different key requirements on 3D city model visualization: Photorealistic visualization to give an intuitive impression of existing or planned environments (left) and abstract visualization to encode thematic information (right).

of an urban area. While in 2D GIS applications exploration and analysis of thematic spatial-related objects and associated thematic information is a common practice, it is still a problem in the case of large-scale 3D city models. The thematic 3D visualization in Figure 1 right, for example, has been manually designed and created by CAD tools and facility management tools. There is no possibility to interact or directly reconfigure the information display.

Our work has been primarily motivated by the requirements of visualization in the context of urban planning and development. For example, in public participation processes such as public discussions between city planners, political decision makers, and the public, 3D city models can be applied

a) to communicate ideas, e.g., where buildings and streets shall be removed, constructed or modified;
b) to discuss feasibility and consequences of certain plan ideas, e.g., how many people would be affected or which housing companies would be involved if a plan would be implemented.

Classical media such as 2D maps or 3D maps do not provide sufficient support for decision-making because they can only show prepared views. An effective tool for decision-support should permit dynamic adaptation of thematic views to the currently focused aspects of a discussion. That is, there are generally more information dimensions than dimensions currently visualized, and we cannot generally assume that specific dimensions are more important than others, in particular, if experts and non-exports want to explore parts of the information space for the first time. In the case of large-scale data sets, we furthermore need to be able to vary the scales in the visualization to support both locally and globally focused investigations.

As a basis for decision-support tools and other applications requiring insight to thematic data, we developed a system for visual data mining in large-scale 3D city models. As stated by Eidenberger [7] “The main aspect of visual data mining is allowing direct communication of the user with the data space through flexible and easy to understand user interfaces". The key characteristics of our system are interactivity and flexibility. As Müller and Schumann point out [9], “Interaction is crucial for effective visual data mining. The data analyst [..] must be able to interact with the presented data and to change the visualizations and the mining parameters”. For this, we support navigating in the virtual space as well as controlling the mapping of thematic information to the city model’s appearance in real-time (Figure 2). Photorealistic detail information such as photo-textures for building façades are not excluded by our system but photorealistic views represent only one of several possible views on an underlying data set.

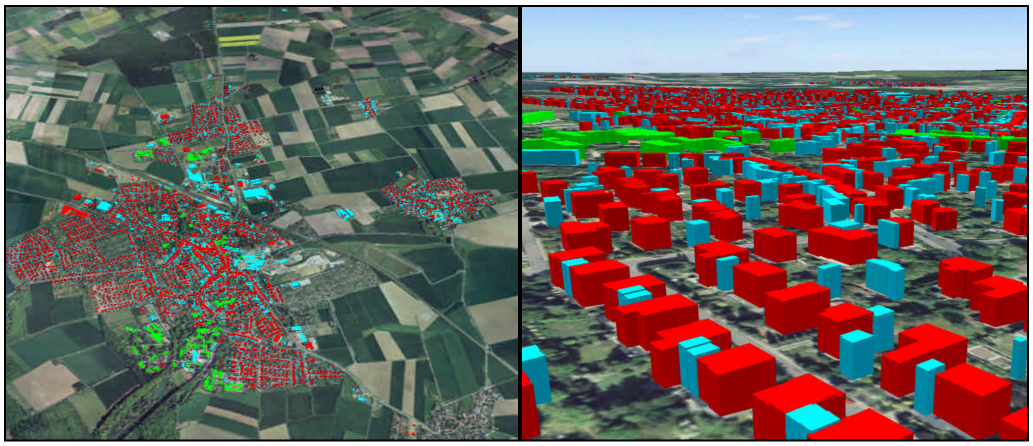
Figure 2: Example of thematic visualization. The building usage has been mapped by the color of each building (Aerial images, terrain model, and city model: ALK, LGN).

DYNAMIC 3D CITY MODELS FOR VISUAL DATA MINING

In this section, we explain the fundamental capabilities of a dynamic 3D city model for visual data mining. The subsequent sections describe the underlying data structures and the rendering technique. First, we specify the way in which our system represents buildings to support thematic visualization. The following building model does only represent an interface between the source data of a city model and our system for visual data mining. It is not intended to provide a complete data model for city model specification.

Modeling Buildings

3D city models are composed of different kinds of objects, including digital terrain models, terrain textures, 3D building models, 3D vegetation models, and 3D graphics objects. Here, we concentrate on 3D buildings as main components through which thematic information should be visualized. Each building objects consists of two components: A geometry description and a building-related attribute table.

The geometry description contains different parts, each of which can be managed separately for information display. The standard parts include *façade*, *roof*, and *unknown*. The *unknown* category is used for buildings whose source specification does not provide the required information to separate façade geometry and roof geometry, e.g., if the whole building geometry is defined by a single set of polygonal faces. Up to now, we observed that the standard categories were sufficient for the previously available city models. Future specification forms for city models, such as Smart Buildings [3] or CityGML [8], however, will provide a more fine-grained categorization than just into *façade* and *roof*.

The attribute table of a building is a set of key-value pairs. Keys are specified by character strings. Values can be character strings as well, but also floating point, integer, or Boolean values, where all values for a certain key must be of equal type. The attribute table is the representation of thematic information within the 3D city model. These tables are created by filtering, importing, and merging one or more data sets into the 3D city model. The reason for the table is that we require a direct and efficient access to thematic information that potentially has an impact on the appearance of building models.

Each city model that provides fully or partially the information described above can be used as source data for our system. As a minimum, the source data must allow us to identify single buildings within the model. An example for an unsuitable city model is a VRML file containing the whole city model as a single geometric description. Appropriate city model specifications that can currently be directly imported are:

- Smart Buildings based on XML
- CityGML, currently an OGC discussion paper for city model specifications
- 3D-Studio MAX object files (.3ds)
- ESRI Shapefiles with 2-dimensional footprint polygons and height values for each building
- ESRI Shapefiles containing an explicit geometric description of each building in the form of boundary polygons.

The building model provides the data basis for the visual data mining system. Each categorized geometric part of each building forms an own entity that can be accessed separately and can therefore be changed independently in its appearance.

Visual Data Mining

To be useful for visual data mining, 3D city model visualization must fulfill two fundamental requirements:

- It must provide means to make non-graphics building information visible, e.g., year of construction, vacancy, state of repair, or the importance of buildings.
- Changing the appearance of buildings must be possible interactively.

To meet the first requirement, we consider the graphics variables available for building models, which include:

- building height;
- façade material, color, and texture;
- façade elements such as doors and windows;
- roof material, color, texture, and type.

These variables can be modified according to thematic data but are frequently needed to support orientation within a city. By integrating different rendering techniques, we obtain additional graphics variables that have no equivalent in photorealistic presentations and, therefore, can easily be perceived as variables for abstract thematic data (Figure 3). Examples include:

- Edge styles applied to facades and roofs [4];
- Transparency of buildings;
- Highlights using haloing effects.

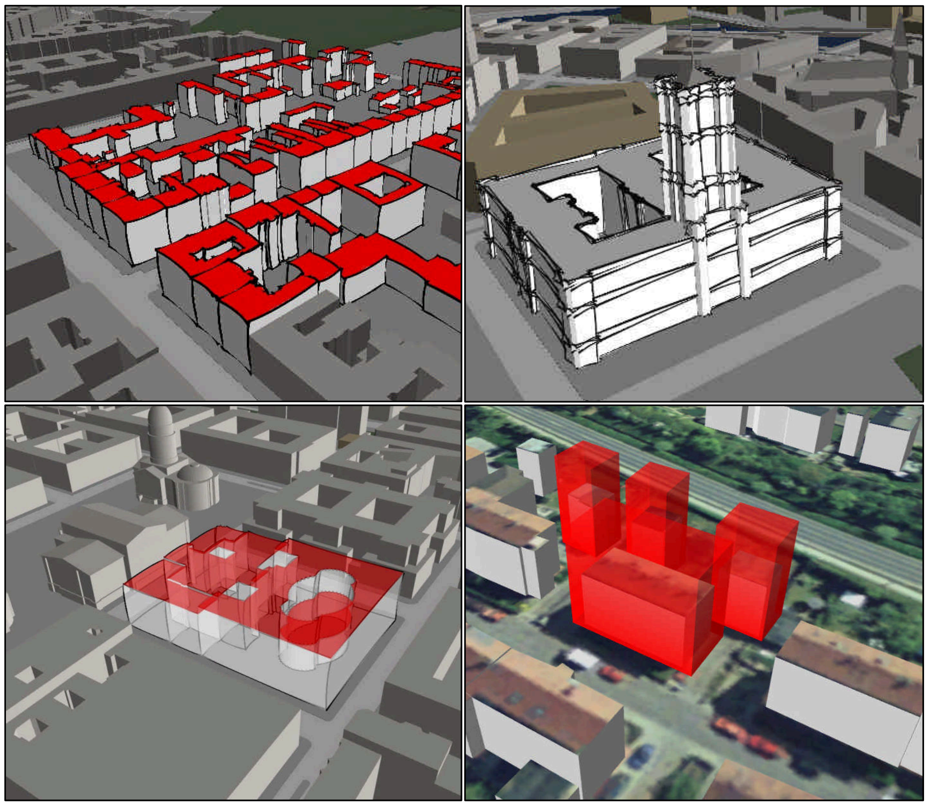

Figure 3: Enhancing the scope of graphics variables by different rendering techniques such as varying edge styles (top), building transparence (bottom left), or halo boxes around buildings (bottom right).

For instance, we can map the status of buildings as planned, existing, or torn off by different edge styles to outline the walls of a building. This kind of graphics variables allow for illustrative depictions known from many classical presentations in urban planning and architecture [4].

To meet the second requirement, visual data mining requires the functionality to interact with a 3D city model in three different ways:

1. Explicit selection: The user can select one or more individual buildings explicitly.
2. Spatial selection: The user can select a group of buildings within a certain area by drawing a polygon onto the terrain.
3. Rule-based selection: The user can specify a selection of buildings by defining a filter condition based on the attribute table of each building. All buildings whose attribute table meet the given filter condition are selected for editing.

A group of selected buildings can be edited simultaneously, either by manipulating their appearance or by changing the attribute table. As a shortcut of rule-based selection and editing, the system also allows for the definition of a color legend: For a certain key of the attribute table, an individual color can be mapped to each occurring value for the key.

SCENE GRAPHS FOR DYNAMIC 3D CITY MODELS

In this section, we explain the internal representation to specify the appearance of a city model such that it can be efficiently and flexibly configured in real-time. The choice of the representation is primarily based on two considerations:

- The representation must not restrict the number of applicable rendering techniques. The integration of new rendering techniques should be facilitated as much as possible.
- The representation should be as compact as possible to work efficiently for large-scale city models.

A representation that meets the first condition is well known in computer graphics: the scene graph. Several open-source graphics libraries support scene graphs, and the implementation of the majority of existing 3D graphics applications is based on them. Each geometry part of each building along with its related appearance attributes could be inserted as individual shapes into the scene graph. For large-scale 3D city models, however, this would result in an extremely large number of scene graph nodes, and real-time rendering would hardly be possibly anymore. For instance, a city model containing 60,000 buildings would be represented by 120,000 leaf nodes (one for roof, one for walls) containing approximately the same number of graphics attributes objects. Even on modern PCs, scene graphs of this scale cannot be evaluated sufficiently fast due to the limited CPU-GPU bandwidth and the limited graphics processing capabilities.

We have to represent a large-scale 3D city model in a more compact way to facilitate a dynamic mapping of thematic data onto its components. In our approach, we represent the geometry of the whole 3D city model by a single shape node of the scene graph. To configure the appearance of the model, we introduce scene graph attributes that control the way in which this geometry is rendered:

- Attributes to define *visual groups*, i.e., groups of buildings sharing common appearance properties. These attributes can be used, e.g., to specify a roof color for all buildings of a certain owner.
- Attributes to manipulate the way in which the system renders 3D representations of buildings.

The additional scene graph attributes have been implemented as an extension to an object-oriented scene graph system [5]. Figure 4 provides an overview of the most important classes of our scene-graph extension. The base class *Attribute* represents a common base class of attributes of the scene-graph system and is not part of our extension. The remaining classes are explained in the subsequent sections.

Forming visual groups

Forming visual groups of buildings sharing common appearance properties is provided by the attribute class *BuildingGrouping*. A BuildingGrouping consists of two parts:

- A *BuildingMapping M,* which defines a mapping from the set off all given building parts of a city model to the set {0,..,k-1} for a constant value k. M defines a subdivision of the complete set of building parts into k groups.
- A sequence of scene graph attributes for each group i in the set {0,..,k-1}. The scene graph attributes of a group are applied to all buildings of the group. BuildingGroupings must not be nested, i.e., a BuildingGrouping cannot contain another one.

Concrete BuildingMappings are implemented as derivatives of the BuildingMapping class. Examples of BuildingMappings are:

- *AttributeTableKeyMapping*: Given a key K of the attribute table and a set of possible values for K, the AttributeTableKeyMapping chooses the group number for a building according to the value that is assigned to K in the attribute table of the building. For instance, an AttributeTableKeyMapping can be used to group buildings of equal roof type. The group number 0 is reserved for all buildings that define no value or an unexpected value for K.
- *RangeMapping*: Given a key K for which numerical values are defined, and a sequence of values $-8=s_0<s_1<..<s_n<s_{n+1}=8$, a RangeMapping defines a group of buildings for each defined interval. If the attribute table of a building defines a value $s_i<v<s_{i+1}$ the building is assigned to group number i+1. The group number 0 is reserved for all buildings for which no numerical value could be found for K. A RangeMapping can be used, e.g., to define different groups based on the year of construction, i.e., to define groups such as “before 1910”, “1910-1950”, “1950-2000”, “2000 or later”.

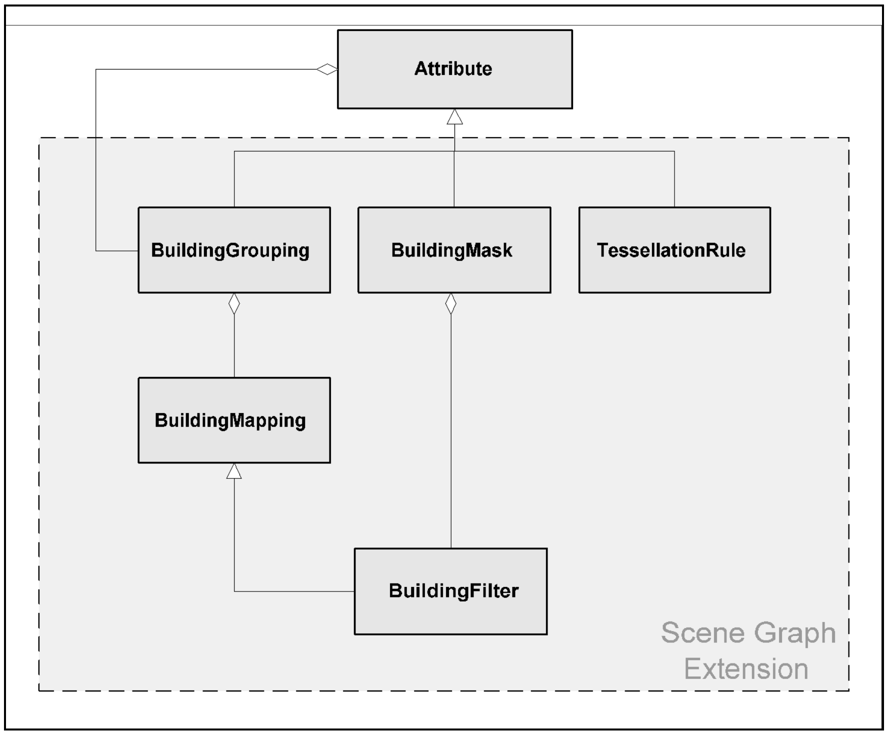

Figure 4: Overview of the central scene-graph extension classes used in our system.

BuildingMappings defining only two groups form an important special case. To simplify the specification of this type of mappings, the class BuildingMapping is specialized by the class *BuildingFilter*. A BuildingFilter is an object that represents a certain condition, i.e., it maps each building part to a Boolean value. Building Filters can be used to define certain selections of buildings, such as in the following examples:

- The *CategoryFilter* accepts all building parts of a certain category, e.g., only building roofs.
- The *AttributeTableCondition* defines a constraint for the value that is defined for a certain key *K* by the attribute table of a building. The AttributeTableCondition defines an *operator* and an *operand*. For numerical values, the possible operators are equal, not equal, less or equal, etc. For string values the possible operators are equal, not equal, contains, and regular expression. The operand is either a numerical value or a string value. In the case of a regular expression, the operand defines the search string.
- An *ExplicitBuildingFilter* defines an explicitly defined set of building parts. This is necessary, e.g., to select a certain set of buildings per mouse.
- The *BoundingPolyonFilter* defines a polygonal area and accepts all buildings within this area.
- A *CombinedFilter* defines a logical expression formed by other filters and logical operators such as AND, OR, NOT, or XOR.

Theoretically, the explicit building filter could replace all possible filters. The drawbacks of explicit filters are that the set of explicit building references can become very large. In addition, the filter object does not provide any information about the actual filter criterion.

The scene graph may contain multiple BuildingGroupings applied to a single city model. For each building part, each BuildingMapping may contribute one or more appearance attributes, dependent on the groups that contain the building part. Figure 5 shows an example: The roofs of all buildings at a certain street are rendered in blue. In addition, a certain explicitly selected set of buildings is colored in red. The intersection of both sets is rendered with blue roof and red facades.

Manipulation of the 3D Building Geometry Creation

In addition to the definition of visual groups, we have to control the way in which our system converts the geometric description of each building part, e.g., a footprint polygon and an additional

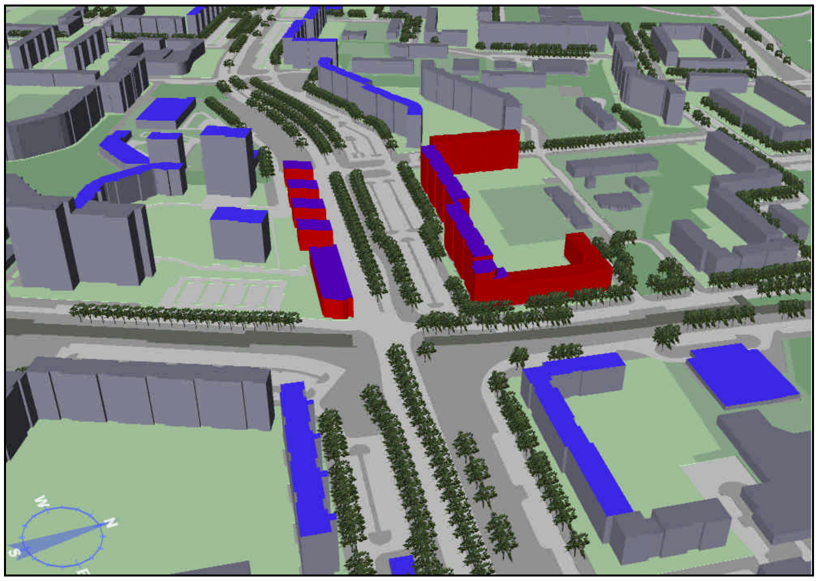

Figure 5: Example of two simultaneously active BuildingGroupings. Blue roofs indicate buildings belonging to a certain street. In addition, a set of explicitly chosen buildings is colored in red.

height value, into a 3D description that can be understood by a 3D rendering subsystem, e.g., Direct3D or OpenGL [11]. A building is usually rendered as a set of triangles. Each triangle vertex is usually represented by a position vector, a color value, and a normal vector. The data related to each vertex are called *per-vertex data*. The explicit control of the 3D geometry creation process is motivated by the following considerations:

- Different rendering techniques require different vertex data. For instance, if a texture is used, additional texture coordinates must be created.
- In some cases, the geometry must be changed completely. For instance, the edge rendering technique in Figure 3 requires an explicit specification of the important edges of a building instead of the building surface.
- Per-vertex data can be individually chosen for each building without the need for a large number of scene graph attributes. For instance, a certain floating-point value of the attribute table can be mapped to a continuous range of colors by manipulation of the per-vertex colors.

The creation of 3D building geometry is manipulated by derivatives of the base class *TessellationRule*. Since the creation process can be time-consuming, it is only involved if changes occur (see Rendering Engine Section). A tessellation rule encapsulates an algorithm that creates a renderable 3D geometric description from a given source specification of a building. In the following, we explain some examples of TessellationRules:

- *RoofTypeTessellationRule*: If the source description of a building provides a footprint polygon, and if the attribute table provides a roof type identifier for a certain key *K*, a RoofTypeTessellationRule creates appropriate roof geometry (Figure 6 right).

- *EdgesTessellationRule*: Given a certain threshold angle α, an EdgeTessellationRule detects all edges whose adjacent triangles form an angle larger than α. The result is an explicit renderable representation of all extracted edges.
- *ScalingTessellationRule*: Setting a vertical scaling factor for a city model cannot be simply achieved by a scaling matrix in the scene graph: Since each building has to be scaled around the plane formed by the intersection of the building and the underlying terrain, each building has to be transformed differently. Hence, we use a ScalingTessellationRule to encode the scaling directly into the vertex positions to avoid the necessity of an individual transform for each building in the scene graph.
- *TextureCoordinatesCreator*: If textures are assigned to certain buildings, additional texture coordinates are needed. The TextureCoordinatesCreator uses the default 3D representation of a

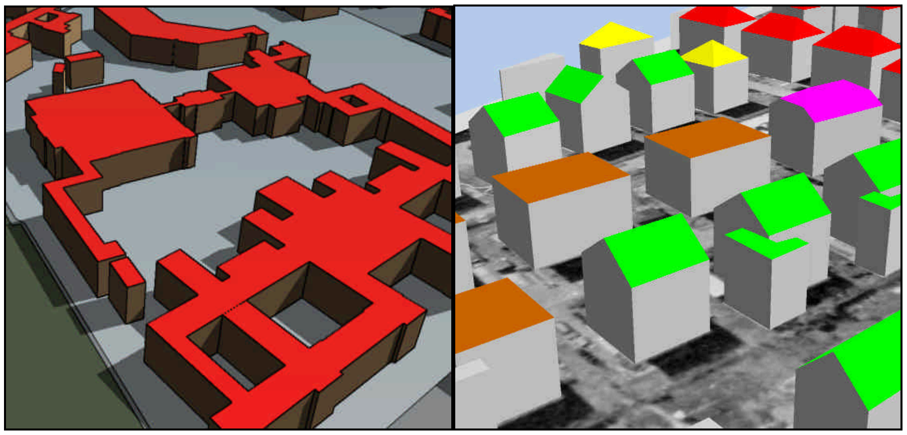
Figure 6: Examples of specific TessellationRules. ShadowVolumeTessellationRule (left) and RoofTypeTessellationRule (right).

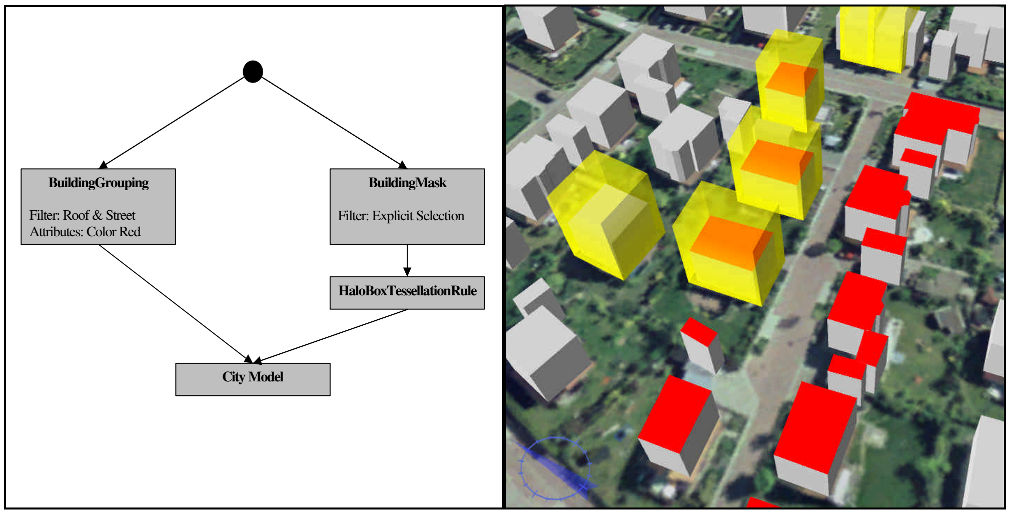

Figure 7: Example scene-graph (left) and a snapshot of the corresponding scene (right). Buildings along a certain street are rendered with red roofs. Explicitly selected buildings are rendered additionally using the HaloBoxTessellationRule.

building and adds additional texture coordinates for each vertex.

- *StaticLighting*: For some applications, it is not required to change the lighting in real-time. In this case, the city model can be rendered faster if the color intensity values for the lighting are calculated only once. The StaticLighting creates a 3D building representation for which the lighting is precomputed in the per-vertex colors.
- *ShadowVolumeTessellationRule*: The shadow technique, illustrated in Figure 6 left, requires the creation of specific shadow-volume geometries for each building. These are provided by the ShadowVolumeTessellationRule.
- *HaloBoxCreator*: Instead of the building surface, a HaloBoxCreator creates a semi-transparent enclosing box for a building (Figure 7 right).
- *ColorMapping*: A ColorMapping provides a continuous mapping of values defined by the attribute table for a certain key *K* to the color of a building.

If a TessellationRule *R* is applied to the whole city model, all buildings of the model are created using the algorithm defined by *R*. TessellationRules can be inserted into BuildingGroupings to restrict their application to certain buildings. The major strength of TessellationRules is the fact that on the one hand we are able to create specialized vertex data for certain rendering techniques, and on the other hand all geometric data can be optimized by the same caching mechanism.

Frequently, multiple tessellation rules have to be applied to a single building. For instance, the halo-box technique in Figure 7 right requires the default representation of a building and additionally the geometry of the halo box. For this, it is possible to insert multiple references to a city model into the scene graph. If additional representations are needed only for certain buildings, these buildings can be filtered by *BuildingMasks*. A BuildingMask is a scene graph attribute that informs the rendering engine to render only a certain subset of the city model. The subset is defined using the BuildingFilters described in the previous section.

Figure 7 left shows an example scene graph: The BuildingGrouping on the left applies the color red to all roofs of buildings along a certain street. The city model is referenced two times in the scene graph and therefore two times rendered. The second rendering is applied, however, only for a small set of explicitly selected buildings. All other buildings are filtered by the BuildingMask on the right. The selected buildings are evaluated using the HaloBoxTessellationRule, which creates the yellow highlighting boxes on the right.

RENDERING ENGINE FOR DYNAMIC 3D CITY MODELS

In this section we describe the basic considerations for rendering a city model with respect to the special scene-graph attributes explained in the previous section. The core of the technique is a caching mechanism that combines the complete city model geometry in a few *geometry batches*. A geometry batch is a large block of geometric primitives - usually triangles - that are all rendered with a single call to the graphics subsystem. The advantage of large geometry batches is the fact that rendering a few large geometry batches is noticeably faster than handling each building part separately [2]. Geometric representations of multiple building parts can be combined, however, only if they have to be rendered with the same appearance attributes. For instance, the edges in Figure 3 must be rendered using a special technique encapsulated in a specific scene graph attribute. Hence, the edge geometry cannot be combined with the triangles describing the building surface.

Although combined geometry batches increase rendering performance, geometry batches should not be chosen too large because of the following reasons:

- Even if only a single building of a batch is changed, the batch needs to be recreated.
- If a batch is completely outside the view frustum, it can be left out from rendering (View-Frustum Culling). Too large batches, however, are possibly never completely outside the view-frustum.
- Smaller batches support incremental updates, which are described later in the corresponding section.

To summarize, the aim of the rendering engine is to combine the geometric representations of all building parts into batches so that:

a) Building parts are only combined if they are to be rendered with equal appearance attributes.
b) The number of geometry batches is substantially smaller than the total number of building parts.
c) Singular batches are small enough to be recreated quickly.
d) Building representations should be combined based on spatial proximity to support view-frustum culling.

Mapping Codes of Buildings

The GroupMapping of each BuildingGrouping defines a group number for each building part. Therefore, the sequence of all BuildingGroupings that are applied to a city model defines a corresponding sequence of group numbers for each building part. We call this sequence the *mapping code* of a building part. Since each BuildingGrouping can define one or more scene-graph attributes for each group, each building part has to be rendered with a combined sequence of scene-graph attributes from all applied BuildingGroupings such as in the example described in the Forming Visual Groups section. Particularly, two building parts have to be rendered with equal scene-graph attributes if they have identical mapping codes. Therefore, we create an own caching data structure, called *cache tree*, for each mapping code for which at least one building part exists. Note that the number of resulting data structures is usually relatively small because simultaneous visualization of more than 3 independent grouping criteria does not lead to understandable visual results.

Structure of a Cache Tree

Each cache tree represents a set of building parts sharing a single mapping code. Hence, to render a city model, all caching data structures are rendered with the appearance attributes corresponding to the mapping code of the cache tree. As stated above, the size of the geometry batches should not become too large. Therefore, a caching data structure does not contain a single geometry batch but a set of geometry batches organized in a spatial data structure. For this, the city model is

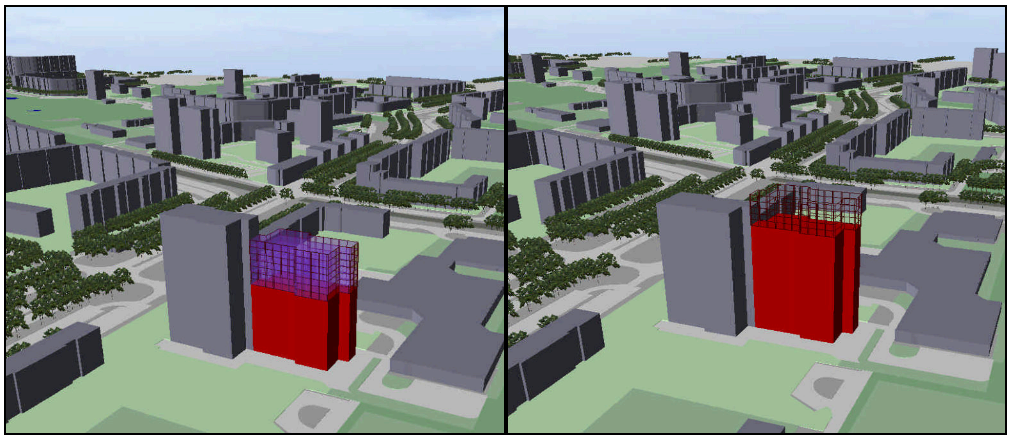

Figure 8: Example application for decision support in urban planning. Floors that are scheduled for removal are indicated in transparent blue, floors that are supposed to be extended are indicated as lines.

subdivided in a quadtree structure [10]. The depth of the quadtree subdivision is chosen in a way that the cells of the leaf nodes contain approximately a fixed predefined number *N* of building representations. N is chosen according to the conditions b) and c) described above. In practice, we chose *N*=1000. All geometric representations contained in a leaf node cell are combined in a geometry batch.

Incremental Updates

If the appearance of the city model is changed, the cache trees have to be appropriately updated. Only geometry batches that are affected by a change are rebuilt. So, many changes of the city model can be done interactively. Some complex updates, however, might require rebuilding several geometry batches. In the extreme case, new geometric representations have to be created for all buildings of a city model. If this would be done in a single frame, the interactivity would be lost. Therefore, our system performs complex updates over multiple consecutive frames. Each time the creation of a geometry batch is finished, this batch is faded in smoothly. If there is a corresponding out-of-date geometry batch, this batch is faded out simultaneously. The smooth fading avoids disturbing popping artifacts. Geometry batches near to the camera are preferred for updating. The user can continue to navigate through the model and even further manipulate the appearance of the city model while the Cache Trees are incrementally updated.

CONCLUSIONS

We described a system for visual data mining in large-scale 3D city models. We have implemented our approach based on the scene-graph system VRS [5] and the graphics library OpenGL [11]. The implementation has been integrated into the geovisualization system LandXplorer [6]. As a case study, we created a decision-supporting tool for an urban planning project of the urban planning office of Berlin (Senatsverwaltung für Stadtentwicklung, Berlin). For this, we extended our system by the specialized feature of visualizing singular floors that are scheduled to be removed or scheduled to be built (Figure 8). To integrate this feature in our system, we only had to implement 3 new TessellationRules: The first one for the transparent parts, the second one for the outlined parts, and the third one for the remaining parts of a building. The caching mechanism described in the paper could be used without the need for further specialization. As a next step, we are going to use our system as a software basis for a public terminal that provides a city model of Berlin along with several information query capabilities. The terminal will be developed in collaboration with the urban planning office of Berlin. As a future work, we plan to enlarge the collection of available rendering techniques of our system. For instance, parameters of different material shaders provide a promising way to provide new graphics variables.

ACKNOWLEDGMENTS

Information on the LandXplorer system is available at www.landxplorer.net. The object-oriented scene-graph system VRS can be found at www.vrs3d.org.

We would like to thank Takis Sgouros and Peter Jürgens, Senate Department of Urban Development, Berlin, for their collaboration and for providing us the city model of Berlin as well as the illustration in Figure 1 right. We are also grateful to the IMK automotive GmbH, Chemnitz, for providing us the city model data of the city of Chemnitz used in Figure 1 left, and to the German environmental foundation Deutsche Bundesstiftung Umwelt for supporting our research within the Lenné3D project (www.lenne3d.de).

REFERENCES

1. Beck, M., Real-time Visualization of Big 3D City Models, International Archives of the Photogrammetry Sensing and Spatial Information Sciences, Vol. XXXIV-5/W10, 2003.

2. Carucci, F. Inside Geometry Instancing, GPU Gems 2, pp. 47-66, Addison-Wesley, 2003.

3. Döllner, J., Buchholz, H., Brodersen, F., Glander, T., Jütterschenke, S., Klimetschek, A., Ad-Hoc Creation of 3D City Models, to appear in: Proceedings of the First International Workshop on Next Generation 3D City Models, EuroSDR, 2005.

4. Döllner, J., Buchholz, H., Nienhaus, M. and Kirsch, F., Illustrative Visualization of 3D City Models, Proceedings of the Conference on Visualization and Data Analysis, pp. 42-51, IS&T/SPIE, 2005.

5. Döllner, J. and Hinrichs, K., A Generic Rendering System. IEEE Transactions on Visualization and Computer Graphics, 8(2), pp. 99-118, April-June 2002, 2002.

6. Döllner, J. and Kersting, O., Dynamic 3D Maps as Visual Interfaces for Spatio-Temporal Data. Proceedings ACM GIS 2000, pp. 115-120, 2000.

7. Eidenberger, H., Visual data mining. SPIE Information Technology and Communication Symposium (Internet Multimedia Management Systems), Philadelphia, USA, 2004.

8. Kolbe, T. H., Gröger, G. and Plümer, L., CityGML – Interoperable Access to 3D City Models. To appear in: Proceedings of the first International Symposium on Geo-Information for Disaster Management, Springer Verlag, 2005.

9. Müller, W. and Schumann, H., Visual Data Mining, NORSIGD Info 2/2002, November 2002, pp. 4-7, 2002.

10. Samet, H., The Quadtree and Related Hierarchical Data Structures. ACM Computing Surveys, 16, pp. 187-260, ACM Press, 1984.

11. Shreiner, D., Woo, M., Neider, J. and Davis, T., OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4, 4th edtition, Addison-Wesley, 2003.

12. Willmott, J., Wright, L.I., Arnold, D.B. and Day, A.M., Rendering of Large and Complex Urban Environments for Real-Time Heritage Reconstructions. Proceedings of the Conference on Virtual Reality, Archaeology, and Cultural Heritage, pp. 111-120, ACM Press, 2001.

AUTHORS INFORMATION

Henrik BUCHHOLZ
buchholz@hpi.uni-potsdam.de
Hasso-Plattner Institute, University of Potsdam

Jürgen DÖLLNER
doellner@hpi.uni-potsdam.de
Hasso-Plattner Institute, University of Potsdam